tree_writer_pkg README

Marc Paterno
CD/CEPA/APS/SLD
v00-03-00

December 19, 2003

Abstract

TreeWriterPkg is a DØ framework package that manages the creation of a Root file containing instances of DØ chunks.

This document contains instructions for users of *TreeWriterPkg*. It describes how to configure *TreeWriterPkg* to save the chunk instances the user wishes to save on the branches named by the user. It also explains how to use the facilities provided by the chunk authors to select the particular chunk instances to be saved.

Chunk developers and maintainers should also read the **edmroot** packages's README file.

Contents

1 Overview

The **tree_writer_pkg** package contains the class *TreeWriterPkg*, a DØ framework mode for the writing of standard DØ chunks to a special Root “tree”. Since this package is a standard DØ framework package, all the normal rules for the operation of a framework program using this package apply. See the framework documentation for an explanation of the general features of framework packages. This document explains only those features specific to *TreeWriterPkg*.

2 The *tnt TTree*

The name of the *TTree* instance created by *TreeWriterPkg* is *tnt*, for ”thumbnail tree”. This tree contains a number of branches, configurable by the user. Each branch contains only one type of chunk. For each event, each branch will contain either zero or one instances of that chunk class.

Strictly speaking, the entries in the branch are not instances of the chunk classes; rather, they are instances of *TClonesArray*. It is the *TClonesArray* that contains either zero objects or one object. Furthermore, the class of the object in the *TClonesArray* instance is also not the chunk class. For a branch that carries instances of *XChunk*, the entry in the *TClonesArray* is actually of *edm::ChunkWrapper<XChunk>*. Because the Root application automatically goes into the *TClonesArray* instance to obtain a pointer to the contained object, users of the *tnt* need only deal with the chunk wrapper class at the Root prompt. The end effect is that to access the member functions of the chunk class stored on the branch, given a branch named ”br”, one must use the chunk wrapper class’s member function `ptr()`[1], which returns a pointer to the underlying instance of the chunk class, on which a member function of the chunk class may be invoked. This is illustrated in Figure 1.

```
root [1] tnt.Draw("br.ptr()->someMemberFunction()")
```

Figure 1: Example of use of the tnt tree at the Root application prompt.

3 Branch Objects

To control what chunk instance goes onto what branch, the user must configure some number of *branch objects*. The creation of branch object

[1]Do not attempt to use the data member of the chunk wrapper class; failure, sometimes spectacular failure, will occur.

classes is described in the README for the package **edmroot**; it is expected that the author of each chunk provides at least one branch object class for his chunk.

Each branch object class is associated with a single chunk class. The purpose of the branch object is to select the matching *instance* of the appropriate class, and to write that instances to the branch with the name specified by the user.

It is up to the user to configure his program with the set of branch objects which are to be available for use. How to do so is explained in Section 4. It is also up to the user to select the subset of the available branch objects which will be active in his program, how those branch objects are configured, and how the mapping of the selected chunks to branch names is to be done. This is explained in Section 5.

4 Program Construction

A program that uses *TreeWriterPkg* is constructed in the same fashion as any other DØ framework program. Consult the DØ framework documentation for general instructions.

The **ctbuild** build system is used to compile and link a program that uses *TreeWriterPkg*. See the **ctbuild** documentation for details on the use of **ctbuild**.

The author (or maintainer) of each chunk class is expected to provide a *branch object registry header* for each branch object class. For a chunk named *XChunk*, it is expected that the branch object class be named *XChunkBO*, and that the registry header file be named `XChunkBO_ref.hpp`. A program that is to write instances of *XChunk* to a branch must include exactly one compilation unit which contains the branch registry header for the branch object class associated with that chunk.

The **ctbuild** system requires at least one source file to trigger the building of a binary target. This source file is the most convenient place to `#include` the branch object reference headers corresponding to the branch objects which the user wishes to have available in the program.

5 Program Configuration

A *TreeWriterPkg* instance is configured with an RCP, as are all DØ framework packages. An example RCP file for *TreeWriterPkg* is shown in Figure 2, and explained below.

Each framework package's RCP must contain a string *PackageName* (line 1), which gives the name of the class which it configures. See the framework documentation for more details.

```
string PackageName = "TreeWriterPkg"   // 1

string branches = ( "tc3", "sc"  )     // 2
string tc3 = ( "TestChunk3BO", "1" )   // 3
string sc = ( "SampleChunkBO" )        // 4

int minimumCountRequired = 3;          // 5

string rootfilename = "test.root"      // 6
```

Figure 2: A sample RCP for configurating *TreeWriterPkg*.

Each *TreeWriterPkg* RCP must contain a vector of strings named *branches* (line 2). This tells the *TreeWriterPkg* the number of branches which are to be written, and their names. It is illegal for a name to be the empty string. Other limitations on the branch names may be imposed by Root; see the current documentation for your version of Root to be sure. In general, short alphanumeric names without embedded punctuation or spaces are safe.

For each string in the vector *branches*, there must be an addtional string with a name that matches that string; in the example in Figure 2, there are two such entries, *tc3* (line 3) and *sc* (line 4). Each of these vectors must contain at least one string.

The first string in the vector is the name of the branch object class that is used to fill the branch. This is how the connection is made from the name of the branch to the type of chunk object to be put on the branch. The remaining (zero or more) strings are passed to the branch object at the time of its construction, as configuration parameters. In addition, the entire package RCP is also passed to *each* branch object at construction time. The documentation for each branch object[2] tells how the strings are to be interpreted, or what additional parameters are to be read from the framework RCP.

Line 5 shows an example of an additional parameter to be used by some branch object during initialization. Note that it is not possible to determine which branch object will need this parameter – perhaps several will. Comments in the RCP can help, as can a naming scheme that prefixes each name with the name of the branch object class. A poor name was chosen in this example to demonstrate the lack of clarity that is caused by user of a poor name. Do not emulate this example!

[2]They all have documentation, don't they?